IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: Gansha WU et al.	Art Unit: 2183
Appl. No. 10/813,599 Confirmation No. 4361	Examiner: Ryan Paul Fiegle
Filed: March 31, 2004	Atty. Docket No. 42339-198432
For: STACK CACHING USING CODE SHARING	Customer No. 26694 PATENT TRADEMARK OFFICE

Amendment and Reply Under 37 C.F.R. §§ 1.111 and 1.121

Honorable Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

In reply to the Non-final Office Action (Office Action) dated **May 15, 2006** (U.S. Patent and Trademark Office Paper No. 532006), Applicants submit the following Amendment and Reply.

It is not believed that extensions of time or fees for net addition of claims are required beyond those that may otherwise be provided for in documents accompanying this paper. However, if additional extensions of time are needed to prevent abandonment of this application, then such extensions of time are hereby petitioned for under 37 C.F.R. § 1.136(a). Any fees required therefor (including fees for net addition of claims), and any other fee deficiency, are hereby authorized to be charged, or any overpayments credited, to our Deposit Account No. 22-0261.

Amendments

Please amend the above-identified application as follows:

Amendments to the Specification begin on page 3 of this paper.

Amendments to the Claims are reflected in the listing of claims which begins on page 4 of this paper.

Remarks begin at page 11 of this paper.

Amendments to the Specification:

Please change the title from "STACK CACHING USING CODE SHARING" to the following amended title:

"CODE INTERPRETATION USING STACK STATE INFORMATION".

Please replace paragraph [0042] with the following rewritten paragraph:

[0042] Computer 700, in an exemplary embodiment, may comprise a central processing unit (CPU) or processor 704, which may be coupled to a bus 702. Processor 704 may, e.g., access main memory 706 via bus 702. Computer 700 may be coupled to an Input/Output (I/O) subsystem such as, e.g., a network interface card (NIC) 722, or a modem 724 for access to network 726. Computer 700 may also be coupled to a secondary memory 708 directly via bus 702, or via main memory 706, for example. Secondary memory 708 may include, e.g., a disk storage unit 710 or other storage medium. Exemplary disk storage units 710 may include, but are not limited to, a magnetic storage device such as, e.g., a hard disk, an optical storage device such as, e.g., a write once read many (WORM) drive, or a compact disc (CD), or a magneto optical device. Another type of secondary memory 708 may include a removable disk storage device 712, which can be used in conjunction with a removable storage medium 714, such as, e.g. a CD-ROM, or a floppy diskette. In general, the disk storage unit 710 may store an application program for operating the computer system referred to commonly as an operating system. The disk storage unit 710 may also store documents of a database (not shown). The computer 700 may interact with the I/O subsystems and disk storage unit 710 via bus 702. Such main memory, secondary memory, disk storage units, and removable disk storage devices are all

non-limiting examples of what may be termed, "machine accessible media." The bus 702 may also be coupled to a display 720 for output, and input devices such as, but not limited to, a keyboard 718 and a mouse or other pointing/selection device 716.

Amendments to the Claims:

This listing of claims will replace all prior versions, and listings, of claims in the application.

Listing of Claims

1. (Original) A method to execute an instruction on an operand stack, the method comprising:

performing a stack-state-aware translation of the instruction to threaded code to determine an operand stack state for the instruction;

dispatching the instruction according to the operand stack state for the instruction; and

executing the instruction.

2. (Original) The method according to claim 1, said performing comprising:

determining a number of operands on the operand stack before the instruction is executed;

determining a number of operands on the operand stack after the instruction is executed based on a number of operands that the instruction consumes and a number of operands that the instruction produces; and

inferring a number of shift operations required after execution of the instruction to maintain top-of-stack elements.

3. (Original) The method according to claim 2, wherein the number of shift operations required after execution of the instruction is based on the number of operands on the operand stack before the instruction is executed and the number of operands on the operand stack after the instruction is executed.

4. (Original) The method according to claim 2, wherein the number of shift operations required after execution of the instruction is inferred based on a static lookup table.

5. (Original) The method according to claim 1, wherein the operand stack is a mixed-register stack.

6. (Original) The method according to claim 1, wherein the operand stack state comprises a number of shift operations to maintain top-of-stack elements of the operand stack after the execution of the instruction.

7. (Original) The method according to claim 6, wherein the top-of-stack elements comprise a register stack.

8. (Original) The method according to claim 1, further comprising:

refilling the operand stack.

9. (Original) A system comprising:

an operand stack to execute an instruction; and

an interpreter to determine a state of the operand stack, translate the instruction into threaded code, and dispatch the instruction based on the state of the operand stack.

10. (Original) The system according to claim 9, wherein the operand stack is a mixed stack comprising a register stack and a memory stack.

11. (Original) The system according to claim 10, wherein the register stack comprises at least one register to hold at least one respective top element of the stack and the memory stack comprises a contiguous memory region to hold the remaining elements of the operand stack.

12. (Original) A machine accessible medium containing program instructions that, when executed by a processor, cause the processor to perform a series of operations comprising:

translating a virtual machine instruction into threaded code based on an operand stack state of the virtual machine instruction;

dispatching the virtual machine instruction according to the operand stack state; and

executing the instruction.

13. (Original) The machine accessible medium according to claim 12, wherein the threaded code is based on an entry point into shared execution code.

14. (Original) The machine accessible medium according to claim 12, further containing program instructions that, when executed by the processor cause the processor to perform further operations comprising:

determining a number of operands that are present on an operand stack at a time before the virtual machine instruction is executed;

determining a number of operands that are present on the operand stack at a time after the virtual machine instruction is executed; and

inferring a number of shift operations required to maintain top-of-stack elements after the virtual machine instruction is executed.

15. (Original) The machine accessible medium according to claim 13, wherein the wherein the number of shift operations required after execution of the instruction is based on the number of operands present on the operand stack at a time before the instruction is executed and the number of operands present on the operand stack at a time after the instruction is executed.

16. (Original) The machine accessible medium according to claim 13, wherein the number of shift operations required after execution of the instruction is inferred based on a static lookup table.

17. (Original) The machine accessible medium according to claim 12, wherein the operand

stack state comprises a number of shift operations to maintain top-of-stack elements of an operand stack after execution of the virtual machine instruction.

18. (Original) The machine accessible medium according to claim 17, wherein the top-of-stack elements comprise a register stack.

19. (Original) The machine accessible medium according to claim 12, further containing program instructions that, when executed by the processor cause the processor to perform further operations comprising:

execute a number of shift operations to replace top-of-stack elements to an operand stack.

20. (Original) The machine accessible medium according to claim 19, wherein the number of shift operations is based on a number of elements on the operand stack that are consumed by the virtual machine instruction and a number of elements that are produced by the virtual machine instruction.

21. (New) The method according to Claim 1, wherein said performing a stack-state-aware translation of the instruction comprises:

determining an entry point into shared execution code based on the stack state.

22. (New) The system according to Claim 9, wherein said interpreter is further to determine an entry point into shared execution code based on the stack state.

23. (New) The machine accessible medium according to Claim 12, wherein said translating a virtual machine instruction into threaded code includes determining an entry point into shared execution code based on the stack state.